Balance dinámico de carga en sorting paralelo

Naiouf Marcelo[1], De Giusti Laura[2], Chichizola Franco[3], De Giusti Armando[4].

III-LIDI – Instituto de Investigación en Informática LIDI.
Facultad de Informática. Universidad Nacional de La Plata.

RESUMEN

El Sorting (ordenación) es una de las operaciones más comunes e importantes que se realizan en una computadora. Numerosos algoritmos requieren que los datos (numéricos o no numéricos) se encuentren ordenados para poder accederlos de manera más eficiente.

Uno de los problemas de los algoritmos de sorting es cuando la secuencia de datos es muy grande. Los mejores algoritmos secuenciales tienen tiempos de orden O(*n x Log n*) donde *n* es el número de datos. La solución al tiempo de procesamiento creciente con *n* ha sido la paralelización de los algoritmos de ordenación, utilizando varios procesadores.

La utilización de múltiples procesadores trabajando sobre subsecuencias del total *n* de datos puede alcanzar un rendimiento cercano al óptimo (speedup = n), pero alcanzar este óptimo es dificultoso en arquitecturas reales.

Uno de los ejes para alcanzar una perfomance óptima en la ordenación paralela es lograr un balance en el trabajo a realizar por cada procesador. Nótese que el trabajo no depende solo de la cantidad de datos de cada subsecuencia, sino también del desorden parcial de la misma, y de la potencia de cómputo de cada procesador.

Este trabajo desarrolla una técnica de redistribución dinámica de la carga de datos a partir de la *predicción* del trabajo a realizar por cada procesador, permitiendo así una carga balanceada del trabajo entre los diferentes procesos. Se demuestra que el método tiende a alcanzar el óptimo teórico en la performance del algoritmo paralelo.

Palabras claves: Ordenación de Datos – Paralelización de Algoritmos – Predicción de perfomance – Balance de carga – Redistribución dinámica.

[1] *mnaiouf@lidi.info.unlp.edu.ar- Profesor Titular UNLP.*
[2] *ldgiusti@lidi.info.unlp.edu.ar- Becaria de Perfeccionamiento UNLP – JTP semi dedicación.*
[3] *francoch@lidi.info.unlp.edu.ar- Becario de Iniciación UNLP – JTP.*
[4] *degiusti@lidi.info.unlp.edu.ar- Profesor Titular UNLP.Ivestigador Principal del CONICET.*

1. Introducción

La evolución del procesamiento hacia el paralelismo ha sido creciente, prácticamente desde el inicio mismo de las computadoras digitales. Los ejes que han impulsado los temas de concurrencia en software y multiprocesamiento en hardware son múltiples, pero se pueden mencionar dos:

- La necesidad de reducir los tiempos de procesamiento de grandes volúmenes de datos (problemas matemáticos, modelos, grandes bases de datos, imágenes, sistemas expertos, biotecnología, etc.).
- El procesamiento de información (datos, señales) en tiempo real para la toma de decisiones tanto en ambientes administrativos como industriales (robótica, industria militar, sistemas multimediales en tiempo real, georeferenciación, reconocimiento de patrones).

Esta evolución conduce a un gran esfuerzo por transformar el procesamiento secuencial en paralelo, buscando reducir los tiempos de ejecución de procesos y de respuesta a eventos del mundo real.

Una de las operaciones que usualmente se requieren como parte de la solución de otros algoritmos más complejos es el "sorting" u ordenación de una secuencia de datos para poder, por ejemplo, acceder a los datos de manera más eficiente. [KUM94]

El problema que surge es el tiempo requerido para lograr este fin cuando la secuencia de datos es muy grande. Los mejores algoritmos secuenciales tienen tiempos de orden O(n x Log n) donde n es el número de datos. Algunos métodos de ordenación son los de selección, intercambio o burbuja (con y sin centinela), inserción, mergesort y quicksort, entre otros. [KNU73][LAN]

La solución al tiempo de procesamiento creciente con *n* ha sido la paralelización de los algoritmos de ordenación. Dada una secuencia de *n* datos, se distribuyen estos entre los distintos procesadores donde son ordenados, para luego mezclarlos, logrando así ordenar la secuencia completa.
La utilización de múltiples procesadores trabajando sobre subsecuencias del total *n* de datos puede tener un rendimiento óptimo. Para alcanzarlo el trabajo que debe realizar cada uno de los procesadores debe ser balanceado. [MIN03] [COL98][XIA93] [AMA96]

Una forma de lograr balancear la carga de los distintos procesos es la de distribuir dinámicamente los datos entre ellos. Para esto se distribuye una parte de los datos entre las distintas tareas, se realiza un porcentaje del trabajo en paralelo, y se estima el trabajo que resta realizar, y a partir de esta predicción se distribuye el resto de los datos buscando balancear el trabajo de todos los procesos.

2. Descripción del trabajo

En este trabajo se analizó el método de intercambio con centinela, el cual realiza una serie de pasadas comparando en cada una de ellas todos los pares de datos adyacentes, e intercambiándolos en caso de que estén desordenados [KNU73]. El algoritmo termina cuando todos los datos se encuentran ordenados. Al paralelizar el algoritmo, cada proceso ordena su subsecuencia por medio de este método, para finalizar la operación al mezclar las partes ordenadas [LAN][COL98][MIN03].

Para conseguir una buena performance global en la ordenación de secuencias de datos, el objetivo es lograr balancear el trabajo realizado por los diferentes procesos.

Para obtener esta equidad, como se explico anteriormente, se utiliza una redistribución dinámica de los datos a partir del trabajo realizado por cada procesador hasta cierto momento de la ordenación, y la predicción del trabajo restante para cada proceso.

El trabajo consta de cuatro etapas, primero la *Predicción del trabajo en forma secuencial*, segundo la *Paralelización del algoritmo de Sort*, tercero la *Predicción del trabajo en el Modelo Paralelo* a realizar por cada proceso, y por último la *Redistribución o balance dinámico* de los datos restantes entre los procesadores.

2.1. Predicción del trabajo secuencial

En el algoritmo utilizado el trabajo necesario para realizar la ordenación de *n* datos no solo depende de la cantidad de datos (*n*), sino también de la distribución que tengan los mismos dentro de la secuencia. Es por esta razón que resulta muy complejo poder determinar el trabajo y por consiguiente el tiempo necesario para realizar la ordenación.

Teniendo en cuenta que este algoritmo se basa en ejecutar una gran cantidad de comparaciones e intercambios, se considera que el trabajo está dado por una combinación de la cantidad de estas operaciones, expresado en la siguiente fórmula:

$$T = CC + (CI * Co) \quad \text{, donde:}$$

T es el trabajo.
CI es la cantidad de intercambios.
CC es la cantidad de comparaciones.
Co es el coeficiente que indica la relación entre el trabajo necesario para un intercambio y una comparación.

A medida que aumenta la cantidad de iteraciones hechas por el algoritmo el trabajo a realizar en cada vuelta va disminuyendo. Debido a esto cuando se han ejecutado el 5% de las vueltas se puede estimar el trabajo necesario para realizar la ordenación completa de los datos. Esta cantidad de iteraciones representa entre 9.5 y 11 % del trabajo total [DEG04]. Por consiguiente el trabajo se puede estimar por la siguiente fórmula:

$$TE = TR*100 / Cp \quad \text{, donde}$$

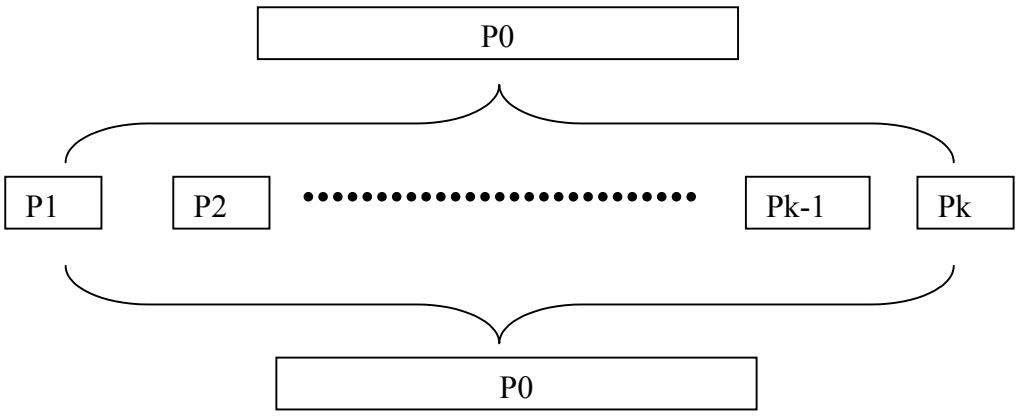
TE es el trabajo total estimado.
TR es el trabajo realizado en el 5% de las pasadas.
Cp es un coeficiente que varía entre 9.5-11, dependiendo de si el TR está dentro de los parámetros normales o no, y representa el porcentaje de trabajo realizado en el 5% de las iteraciones.

2.2. Paralelización del algoritmo de Sort

Existen numerosas técnicas para Sorting Paralelo, una de ellas es el Sorting by Merging, que utilizan el paradigma master-slave y consideran los siguientes pasos:

- Se divide los *n* items en *k* subsecuencias de igual tamaño.
- Cada subsecuencia es enviada a un procesador diferente, el cual la ordena utilizando el método de "Intercambios con Centinela" antes descripto.
- Se realiza la mezcla (Merge) de las *k* subsecuencias ordenadas por cada procesador, obteniendo la secuencia completa ordenada. Esta etapa de Merge se realiza en un solo paso en vez de realizar un merge multifase, debido a que esta última solución requiere más comunicación entre los procesadores afectando a la performance del algoritmo. [DEG04].

En la siguiente figura se visualizan las tres etapas del algoritmo.



De esta manera la performance del algoritmo de ordenación queda determinado por el proceso que más trabajo debe realizar.

2.3. Predicción del trabajo en el modelo paralelo

Por lo explicado anteriormente cada proceso realiza la ordenación de cierta cantidad *k* de datos, y para estimar el trabajo a realizar por cada uno de ellos se utiliza la fórmula detallada en el paso 2.1. Como los procesos están ejecutándose en forma paralela el trabajo considerado para esta etapa está determinado por el del proceso que más trabajo va a realizar.

Por lo tanto para estimar el trabajo total a lo anterior debe sumarsele el trabajo para realizar el merge entre las subsecuencias ordenadas para así obtener la secuencia final.

La estimación está expresada en la siguiente fórmula:

$$T = \max_{i \in [1..k]} (TEi) + M \text{ , donde:}$$

T es el trabajo total.
TEi trabajo estimado para el proceso i por medio de la fórmula explicada en el punto 2.1.
M trabajo para realizar el merge.

2.4. Redistribución o Balance dinámico

A partir de la estimación del trabajo a realizar en cada proceso por medio de la fórmula encontrada en la ***etapa 2.2***, se puede distribuir los datos de manera más adecuada para lograr así equiparar el trabajo de cada proceso.

La redistribución se realiza de la siguiente manera:

- Se reparte un porcentaje de los datos (bloque1) en forma equitativa a cada uno de los procesos, obteniendo así un resto sin distribuir.
- Se estima el trabajo a realizar por cada proceso a partir del realizado en el 5% de las iteraciones, según la formula mencionada en 2.2.
- Se redistribuye el resto de los datos (bloque2) tratando de equiparar el trabajo total que realizará cada proceso (trabajo total es = trabajo (bloque1) + trabajo (bloque2)).
- Cada proceso realiza el merge del bloque1 y el bloque2.
- Se reúnen las subsecuencias ordenadas de cada proceso y se realiza el merge para poder obtener la secuencia final ordenada.

3. Experimentación y Resultados obtenidos

El soporte utilizado para la experimentación fue el lenguaje C con la librería MPI para la comunicación, sobre un cluster de 20 PC's homogéneas (Pentium IV) conectadas a través de una red ethernet de 100 Mbytes.

En este trabajo se implementaron dos algoritmos para comparar el balance de carga contra la del método propuesto de redistribución dinámica. Una breve descripción de los mismos es la siguiente:

- Sorting paralelo sin redistribución (SPSR), en el cual todos los datos son distribuidos en forma equitativa (en cuanto a cantidad de datos) entre todos los procesos.
- Sorting paralelo con redistribución fija (SPRF), en el cual se reparte inicialmente un porcentaje de los datos en forma equitativa (en cuanto a cantidad) y luego se reparte el resto de la misma forma.

Se realizaron numerosas pruebas que incluyeron diferentes tamaños de secuencias a ordenar (50000, 100000, 500000, y 1000000 de datos) como también distintas cantidades de tareas a utilizar (4, 8 y 16 tareas). Además también se varió el porcentaje de datos sin repartir inicialmente (10%, 20%, 30% y 40%).

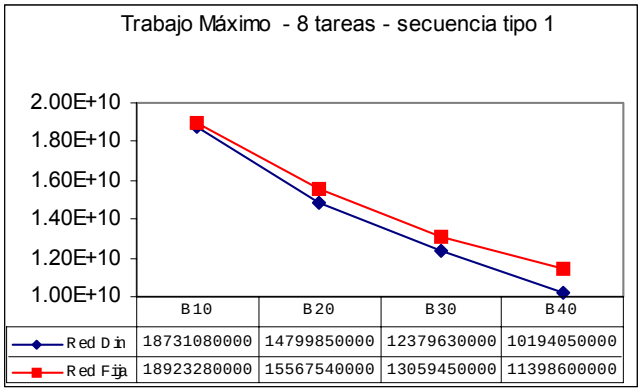
También las pruebas incluyeron distintos tipos de secuencias de acuerdo a la distribución de los datos que se reparten inicialmente:

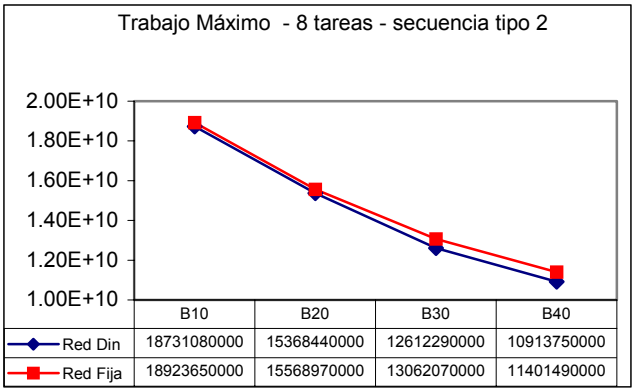
- los datos enviados a una tarea están totalmente invertidos, y el resto son aleatorios (tipo 1).
- los datos enviados a la mitad de las tareas están invertidos y el resto en forma aleatoria (tipo 2).
- todos los datos enviados a las tareas son aleatorios (tipo 3).

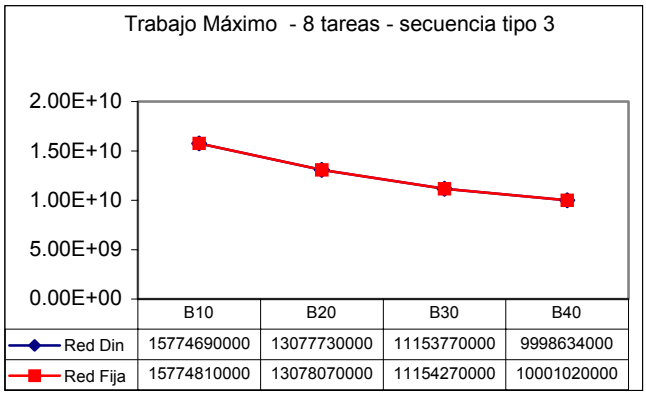
El balance de carga se estableció como la diferencia entre el trabajo máximo y mínimo dividido por el trabajo promedio realizado entre todos los procesos involucrados en la ejecución.

3.1. Trabajo Máximo

Como se mencionó el tiempo de ejecución del algoritmo paralelo se ve influido por la tarea más lenta; en este caso es aquella que debe realizar mayor cantidad de trabajo.
En la siguiente figura se muestra el trabajo máximo realizado por cada uno de los tres algoritmos con 100000 de datos, 8 tareas, utilizando los tres tipos de secuencias mencionadas.







En los gráficos puede verse que el trabajo máximo en el método propuesto siempre es menor al realizado por los otros métodos.
Es importante destacar que a mayor porcentaje de datos no repartidos inicialmente es menor el trabajo máximo alcanzado en el método propuesto, ya que se mejora el ajuste en la redistribución.

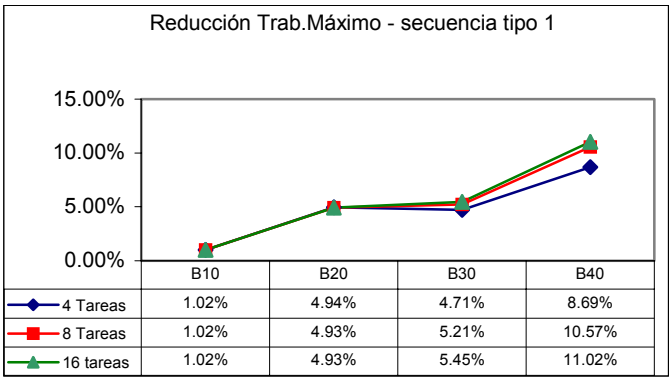
Además se muestra que en las secuencias de tipo 1 la diferencia entre el método propuesto y los otros dos es mayor.

3.2. Porcentaje de Reducción del Trabajo Máximo

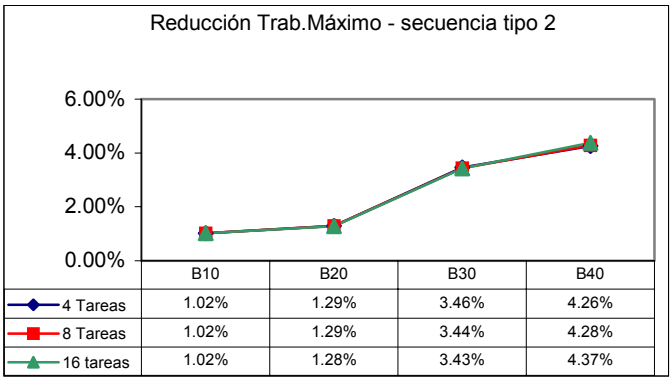
En el siguiente gráfico se muestra la relación de reducción entre los trabajos máximos realizados en los métodos con redistribución dinámica y fija, para 100000 de datos y 8 tareas, utilizando secuencias de los tres tipos. La reducción se obtiene realizando la siguiente cuenta:

$$R = ((TMF - TMD) / TMF) * 100, \text{ donde}$$

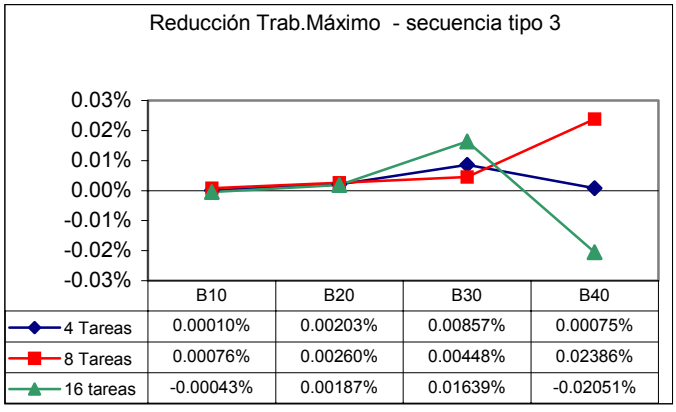
R es el porcentaje de reducción obtenido.
TMF es el trabajo máximo con redistribución fija.
TMD es el trabajo máximo con redistribución dinámica.



	B10	B20	B30	B40
4 Tareas	1.02%	4.94%	4.71%	8.69%
8 Tareas	1.02%	4.93%	5.21%	10.57%
16 tareas	1.02%	4.93%	5.45%	11.02%



	B10	B20	B30	B40
4 Tareas	1.02%	1.29%	3.46%	4.26%
8 Tareas	1.02%	1.29%	3.44%	4.28%
16 tareas	1.02%	1.28%	3.43%	4.37%



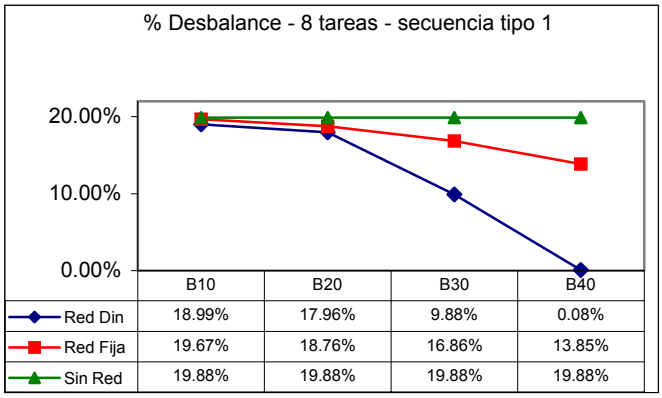
	B10	B20	B30	B40
4 Tareas	0.00010%	0.00203%	0.00857%	0.00075%
8 Tareas	0.00076%	0.00260%	0.00448%	0.02386%
16 tareas	-0.00043%	0.00187%	0.01639%	-0.02051%

En los gráficos anteriores puede observarse que la reducción se da en mayor medida en las secuencias de tipo 1 y en particular cuando las cantidades de datos sin redistribuir representan un 30 y 40 porciento de los datos.
Así también se muestra que cuando las secuencias son del tipo3 la diferencia entre los distintos algoritmos no es tan amplia (notar la diferencia de escala).

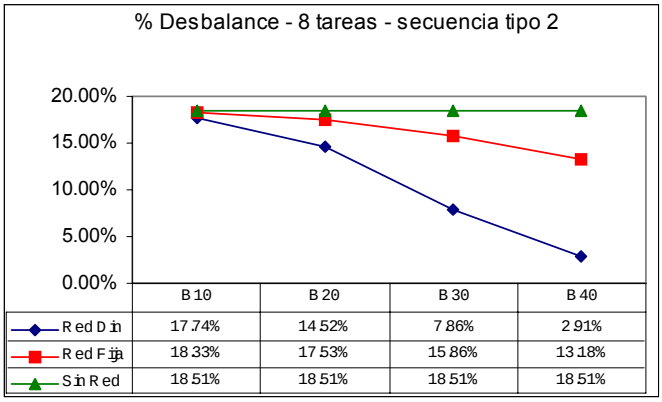
3.3. Balance de carga

Para analizar el balance de carga en el sistema, se presentan la diferencia entre el trabajo máximo y mínimo realizado en cada uno de los algoritmos. Por otro lado también se calcula el porcentaje de desbalance que representa dicha diferencia con respecto al trabajo promedio.

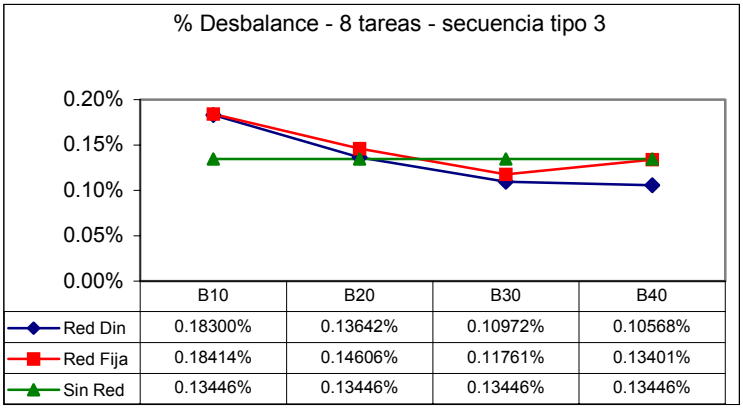
Como aclaración se destaca que el trabajo máximo para cada algoritmo es la cantidad de trabajo del proceso que más trabaja, en forma análoga el trabajo mínimo para cada algoritmo es la cantidad de trabajo del proceso que menos trabajo realiza y el trabajo promedio es el promedio de trabajo entre todos los procesos.



	B10	B20	B30	B40
Red Din	18.99%	17.96%	9.88%	0.08%
Red Fija	19.67%	18.76%	16.86%	13.85%
Sin Red	19.88%	19.88%	19.88%	19.88%



	B 10	B 20	B 30	B 40
Red Din	17.74%	14.52%	7.86%	2.91%
Red Fija	18.33%	17.53%	15.86%	13.18%
Sin Red	18.51%	18.51%	18.51%	18.51%



	B10	B20	B30	B40
Red Din	0.18300%	0.13642%	0.10972%	0.10568%
Red Fija	0.18414%	0.14606%	0.11761%	0.13401%
Sin Red	0.13446%	0.13446%	0.13446%	0.13446%

Los gráficos anteriores muestran como el algoritmo de redistribución dinámica reduce el desbalance de carga, para secuencias de los tres tipos (1,2), esta reducción no es visible para las secuencias de tipo 3, debido a que la distribución de los datos estadísticamente genera un balance de carga adecuado.

4. Conclusiones y Líneas de trabajo futuras

Existe una gran cantidad de técnicas de balance de carga tanto estáticas como dinámicas. Si bien las técnicas estáticas son más sencillas de realizar, no pueden ser utilizadas cuando no se conoce de antemano la carga de trabajo a realizar. En estos casos redistribuyendo la carga en forma dinámica se puede lograr un mejor balance de carga, sin producir un gran overhead.

En este trabajo se logró encontrar una forma de estimar *en forma dinámica* el trabajo a realizar para ordenar una secuencia de datos por medio del método de intercambios con centinela.

Al poder predecir la performance de la ordenación en paralelo a partir del trabajo estimado para cada proceso, se pudo corregir y optimizar la distribución de datos por procesador, mejorando de este modo la performance de la operación completa.

En los graficos puede apreciarse que el método de redistribución dinámica propuesto equilibra mejor la carga de trabajo entre los procesadores, reduciendo el trabajo máximo a realizar y por consiguiente el trabajo final, optimizando así la performance del sistema.

Como trabajo futuro se está tratando de incluir la heterogeneidad de los procesadores como otro factor para determinar la cantidad de trabajo que debe realizar cada procesador.

También se están realizando las modificaciones necesarias al algoritmo para ejecutarlo en diferentes modelos de arquitectura, en particular, arquitecturas de memoria compartida distribuida (SGI Origin2000).

5. Bibliografía

- [AKL89], Akl S, “The Design and Analysis of Parallel Algorithms”, Prentice-Hall, Inc., 1989.
- [AKL97], Akl S, “Parallel Computation. Models and Methods”, Prentice-Hall, Inc., 1997.
- [AMA96], Amato Nancy M., Ravishankar Iyer, Sharad Sundaresan, Yan Wu, "A Comparison of Parallel Sorting Algorithms on Different Architectures", Technical Report No. 98/029, Department of Computer Science. Texas A\&M University, 1996.
- [AND95], Andrews, "Concurrent Programming", Benjamin/Cummings, 1991.
- [BAD02], Bader, B. Moret, P. Sanders, "Experimental Algorithmics From Algorithm Design to Robust and Efficient Software”, Vol. 2547 of Lecture Notes of Computer Science, Algorithm Engineering for Parallel Computation, Springer Verlag,2002.
- [BLE98], Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton and S. J. Smith, M. Zagha, "An Experimental Analysis of Parallel Sorting Algorithms",Theory of Computing Systems, Vol.31 Nº2, 1998.
- [BRI95], Brinch Hansen, P., “Studies in computational science: Parallel Programming Paradigms”, Prentice-Hall, Inc., 1995.
- [BUB97], Bubak, Funika, Moscinski, “Performance Analysis of Parallel Applications under Message Passing Environments”, www.icsr.agh.edu.pl/publications/html/perf_full/, 1997.
- [BUS88], Bustard, Elder, Welsh, "Concurrent Program Structures", Prentice Hall, 1988.
- [CHA88], Chandi K. M., Misra J., “Parallel Program Design. A Foundation”, Addisson Wesley, 1988.
- [COF92], Coffin M, "Parallel programming- A new approach", Prentice Hall, Englewood Cliffs, 1992.
- [COL95], Colección de "IEEE Transactions on Parallel and Distributed Systems", IEEE.
- [COL98], Cole R., "Parallel Merge Sort", Siam Journal of Computing, Vol.17 Nº4, 1998.
- [DEB90], De Blasi Mario, "Computer Architecture", Addison-Wesley, 1990.
- [DEG04], De Giusti Laura, Chichizola Franco. Informe Técnico.
- [GOO00],Goodrich Michael T., "Communication-Efficient Parallel Sorting", Siam Journal on Computing, Vol.29 Nº2, 2000.
- [GUP93], Gupta A., Kumar V., “Performance properties of large scale parallel systems”, Journal of Parallel and Distributed Computing, November 1993.
- [GUS88], Gustafson J., “Reevaluating Amdahl´s Law”, Comm. Of the ACM, Volume 31 (1988), pp. 532-533.
- [HAN92], Hanmao Shi, Jonathan Schaeffer, "Parallel Sorting by Regular Sampling", Journal of Parallel and Distributed Computing Vol.14, Nº 4, 1992.
- [HEE91], Heermann, A. N. Burkitt, "Parallel Algorithms in Computational Science", Springer-Verlag, 1991.
- [HOA85], Hoare, "Communicating Sequential Processes", Pentice-Hall, 1985.
- [HWA93], Hwang, "Advanced Computer Architecture. Parallelism, Scalability, Programmability", McGraw Hill, 1993.
- [KAR92], Karonis N, “Timing Parallel Programs That Use Message Passing”, Journal of Parallel and Distributed Computing, 14, 1992.
- [KNU73], Knuth, "The Art of Computer Programming, Vol 3: Sorting and Searching", Addison-Wesley, 1973.
- [KUM94], Kumar V., Grama A., Gupta A., Karypis G., “Introduction to Parallel Computing. Desing and Analysis of Algorithms”, Benjamin/Cummings, 1994.

- [KUN91], Kung H. T., Sansom R., Schlick S., Steenkiste P., Arnould M., Bitz F. J., Christianson F., Cooper E. C., Menzilcioglu O., Ombres D., and Zill B., “Network-Based Multicomputers: An Emerging Parallel Architecture” Proc. Supercomputing '91, pp. 664-673. IEEE, Albuquerque, Nov. 1991.
- [LAN], Lang, "Sequential and Parallel Sorting Algorithms", http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htm.
- [LUQ95], Luque E, Ripoll A, Cortès A, Margalef T, “A Distributed Diffusion Method for Dynamic Load Balancing on Parallel Computers”, Proceedings of the EUROMICRO Workshop on Parallel and Distributed Processing, IEEE Computer Society, Jan. 1995.
- [MIL98], Miller R., Stout Q. F., “Algorithmic Techniques for Networks of Processors”, CRC Handbook of Algorithms and Theory of Computation, M. J. Atallah, ed, 1998.
- [MIN03], Minsoo Jeon, Dongseung Kim, "Parallel Merge Sort With Load Balancing", International Journal of Parallel Programming, Vol.31 Nº1, 2003.
- [MOR94], Morse F., “Practical Parallel Computing”, AP Professional, 1994.
- [NAI04], Naiouf Marcelo, “Procesamiento paralelo. Balance de carga dinámico en algoritmos de sorting.”, Tesis Doctoral en prensa, 2004.
- [SIM97], Sima D, Fountain T, Kacsuk P, “Advanced Computer Architectures. A Design Space Approach”, Addison Wesley Longman Limited, 1997.
- [STE96], Steenkiste P., “Network-Based Multicomputers: A Practical Supercomputer Architecture”, IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 8, August 1996, pp. 861-875.
- [TIN98], Tinetti F., De Giusti A., "Procesamiento Paralelo. Conceptos de Arquitectura y Algoritmos", Editorial Exacta, 1998.
- [XIA93], Xiaobo Li, Paul Lu, Jonathan Schaeffer, John Shillington, Pok Sze Wong, Hanmao Shi, "On the versatility of Parallel Sorting by Regular Sampling", Parallel Computing Vol.19, 1993.
- [ZOM96], Zomaya A. (ed), “Parallel Computing. Paradigms and Applications”, International Thomson Computer Press, 1996.